システムソフトウェアと
オペレーティング・システム 63－6
（１９９４．３．２２）

メモリ管理を考慮した２レベルスケジューリング

甲斐久淳† 藤木亮介† 福田晃†‡

†：九州大学 工学部 情報工学科
‡：奈良先端科学技術大学院大学

NUMA (Non-Uniform Memory Access) マルチプロセッサを対象とする，メモリ管理を考慮した２レベルスケジューリングについて述べる．本稿ではまず，メモリ管理を考慮した２レベルスケジューラの構造とインタフェースについて述べる．さらにメモリ管理とスケジューリングの協調動作の有効性，および実現方策における基本的な選択肢を評価するために，(1) プロセス空間のローディング方式，(2) フリープロセッサの割り当て方式の２項目についてシミュレーションにより比較・検討する．その結果，メモリマネジャによるプロセス空間のローディングに対しては，スケジューラからのフリープロセッサ数に関する情報提供が，また，スケジューラによるフリープロセッサ割り当てについては，プロセス空間の割り当てに関する情報の提供が効果的であるということが分かった．

Two-level Scheduling in Collaboration with Memory Management

Hisa'aki Kai† Ryosuke Fujiki† Akira Fukuda†‡

†: Department of Computer Science and Communication Engineering
Faculty of Engineering, Kyushu University
‡: Nara Institute of Science and Technology
Correspondence: {kai, fujiki, fukuda}@csce.kyushu-u.ac.jp

Two-level scheduling strategy in collaboration with memory management for NUMA (Non-Uniform Memory Access) multiprocessors is discussed. In this paper, we describe about the structure of two-level scheduler and interfaces for it. Through simulation experiments, we evaluate the performance of the alternatives from the viewpoints of the following: 1) process loading and 2) free-processor allocation. For process loading, information about the number of free processors is effective. For free-processor allocation, information about allocation of address space of process is effective.

1 まえがき

マルチプログラミング環境下のマルチプロセッサのスケジューリング方式は，時分割方式（time-multiplexing）と空間分割方式（space-multiplexing）の2つに大別される [4]．時分割方式は，同一プロセス（本稿では，実行するプログラムをプロセスと呼ぶ）内の全スレッド（並列実行可能な単位，アクティビティ）をプロセッサ群に同時に割り当てて，実行し，タイムスライスを使いきると，次のプロセスを実行する方式である．代表的なものとしてコスケジューリング [5] が挙げられる．時分割方式は，いくつかの利点がある反面，スケジューラボトルネックやキャッシュ活用上の問題点があり，大規模マルチプロセッサには適さないと考えられる．一方，空間分割方式は，プロセッサをグループ化し，グループ単位にプロセスを割り当ててスケジューリングする方式であり，大規模マルチプロセッサのスケジューリング方式として有望な方式である．2レベルスケジューリングは，空間分割方式の1つであり，我々は，その有効性，および2レベルスケジューリング方式における基本的な選択肢を，UMA(Uniform Memory Access)/NUMA(Non-Uniform Memory Access) マルチプロセッサを対象として，シミュレーションにより評価してきた [1, 2, 3]．将来有望な NUMA(Non-Uniform Memory Access) マルチプロセッサを対象としたスケジューリングの研究は，現在までのところあまり行われていないが [6, 7]，今後活性化される分野である．

そもそも，スケジューリング方策は，プロセスのメモリへの割り当て方策と密接な関係があり，両者の間には何らかの協調が必要である [8]．特に，メモリの不均一性を有する NUMA マルチプロセッサにおいては，メモリを含めてスケジューリングを考えることは不可欠であると考えられる．しかし，現在のところ，まだメモリ管理と連動したスケジューリングの研究はほとんど行われていない．

本稿では，マルチプログラミング環境下の NUMA マルチプロセッサ上で，メモリ管理を考慮した2レベルスケジューリングに関する研究の第一歩として，その有効性および各種の基本的な選択肢をシミュレーションにより評価する．

2 構造とインタフェース

2.1 構造

図1に示すように，2レベルのスケジューラとメモリマネジャから構成される．

スケジューラ

スケジューラは上位スケジューラ（グローバル・スケジューラ）と下位スケジューラ（プライベート・スケジューラ）から構成される．グローバル・スケジューラはさらに，プロセッサの動的グルーピングを行なうグループサーバと，フリープロセッサ（どのグループにも属さないプロセッサ）を置くプロセッサ・プールの管理を行うプールサーバから構成される．各プロセスはグループ・サーバによって1つのプロセッサ・グループに割り当てられる．グローバル・スケジューラはシステム内に一つ存在し，カーネル空間で動作する．プライベート・スケジューラは，グループ内プロセッサを用いて対応するプロセス内のスレッドをスケジューリングし，対応するプロセス空間に存在する．

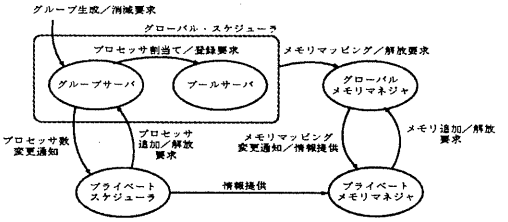


図1 スケジューラとメモリマネジャ

メモリマネジャ

スケジューラと同様に，グローバル・メモリマネジャとプライベート・メモリマネジャから構成される．グローバル・メモリマネジャは，複数のプロセス空間の物理メモリへのグローバルな割り当て／管理を行う．プライベート・メモリマネジャは，プロセス対応に存在し，対応するプロセス空間の物理メモリへの割り当て／管理を行う．

2.2 インタフェース

スケジューラ，メモリマネジャ内の各モジュールが提供するインタフェースを以下に示す (図1)．スケジューラのインタフェースの詳細は文献 [1, 2, 3] を参照して戴きたい．

グループ・サーバ

- **グループ生成要求：** プロセッサ・グループの生成とプロセスへのグループの割り当てを行なう．
- **グループ消滅要求：** 対応するプロセッサ・グループの消滅とプロセッサの解放を行なう．
- **プロセッサ追加要求：** プロセッサ・グループへのプロセッサの追加割り当てを行なう．
- **プロセッサ解放要求：** プロセッサ・グループからプロセッサの解放を行なう．

プール・サーバ

- **プロセッサ割り当て要求：** プロセッサ・グループへのプロセッサの割り当てを行なう．

- **プロセッサ登録要求：** フリープロセッサのプロセッサ・プールへの登録を行なう．

プライベート・スケジューラ

- **プロセッサ数変更通知：** グループに割り当てられているプロセッサ数が変更されたことを通知してもらうためのインタフェース．

グローバル・メモリマネジャ

- **メモリマッピング要求：** プロセス空間の一部または全体を，物理メモリにローディングし，マッピングする．本インタフェースは，プロセッサ・グループ生成後にグローバル・スケジューラによって使用される．
- **メモリマッピング解除要求：** プロセスの終了により，プロセスに割り当てられていた物理メモリを解放する．
- **メモリ追加要求：** ヒープ領域の動的確保，および論理ページの動的マッピング（ページのコピー，および移動）などに伴う物理メモリの確保のために使用される．
- **メモリ解放要求：** ヒープ領域の動的解放，および論理ページの動的マッピングに伴って，不必要となった物理メモリを解放する．

プライベート・メモリマネジャ

- **メモリマッピング変更通知：** グローバル・メモリマネジャの方策などにより，当該プロセス空間の物理メモリへのマッピングが強制的に変更された場合に使用される．
- **情報提供：** プロセス内のスレッドをスケジューリングする時，およびグローバル・メモリマネジャによるメモリ管理に必要な，プロセス内のマッピング情報を提供する．

3 対象シミュレーションモデル

3.1 マルチプロセッサモデル

複数のクラスタからなるNUMAマルチプロセッサシステムを対象とする(図2)．各クラスタは，8個のプロセッサとプロセッサ対応のキャッシュ，1つの物理メモリから構成される．システムは全8クラスタ，計64台のプロセッサから構成される．

3.2 プロセスモデル

プロセスの生成間隔は指数分布に従うものとする．

1つのプロセスは，fork-joinタイプの複数スレッドの同時生成／消滅と逐次実行を2回繰り返すパターンとする(図3)．ジョイン時の同期プリミティブはブロッキングロック(blocking lock)とする．

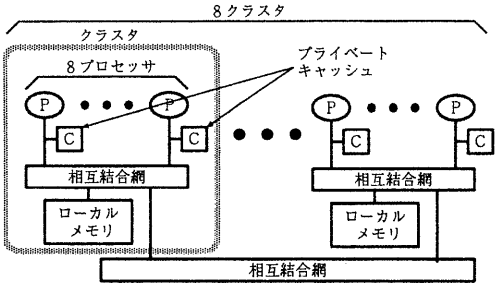


図2 NUMAマルチプロセッサモデル

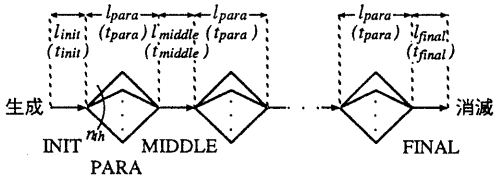


図3 プロセスグラフ

全てのプロセスのアドレス空間は32の論理ページから構成される．各ページは読み出し専用(Read-Only, R-O)または，読み書き(Read-Write, R/W)の属性を持つ．スレッドの実行はアドレス空間内の論理ページの時系列で与えられる．各論理ページに関する実行時間(後で述べるメモリアクセスオーバヘッドを考慮しない値，ページの種類に関わらず一定)を T とし，INIT, MIDDLE, FINAL, PARAフェイズにおける各スレッドの論理ページの時系列長を l_{init}, l_{para}, l_{middle}, l_{final} とする(総称して l_{phase} と表すことにする)．各パラメータの値としは，$T = 3$, $l_{init} = l_{middle} = 4$, $l_{final} = 10$ を固定とし，PARAフェイズにおける各スレッドの論理ページの時系列長 $l_{para} = U(11, 33)$，および，forkで生成されるスレッド数 $n_{th} = U(20, 28)$ をプロセス毎に変化させるものとする．ここで，$U(a, b)$ は a から b までの一様分布関数である．

3.3 オーバヘッド

メモリアクセスオーバヘッド

異なる種類のメモリアクセス時間を持つNUMAマルチプロセッサにとって，メモリアクセスのオーバヘッドはプロセスの実行時間に大きな影響を与える．本論文のシミュレーションではメモリ・アクセスのオーバヘッドをモデル化している．

スレッドの実行は，アクセス属性を持つ論理ページの時系列で与えられる．時系列中の各論理ページに関する実行時間 $t_{page_i}(i = 0, 1, \ldots, l_{phase} - 1)$ は，アク

表1 メモリアクセス・オーバヘッド

論理ページの位置	キャッシュの有無	コピーの有無	t_{page_i}
ローカル	有	—	T
ローカル	無	—	$3T$
リモート	有	無	$5T$
リモート	無	無	$7T$
リモート	有	有	$15+T$
リモート	無	有	$15+3T$

セスする論理ページのキャッシュ内へのキャッシングの有無，主記憶内での位置，および，ページコピーの有無により異なる(表1)．キャッシングについては実際はキャッシュブロック単位で行なわれるが，本論文では簡単のため，プロセッサが直前にアクセスしたページと同一の論理ページをアクセスしているかどうかにより判断している．主記憶内における位置は，プロセス空間のロードやページコピーによりローカルメモリに論理ページがあるかどうかで判断する．ページコピーは，プロセッサがアクセスする読み出し専用ページがリモートメモリにある場合に行なわれる．

スケジューラとメモリマネジャ

シミュレーションではスケジューラ／メモリマネジャの実行によるオーバヘッドをモデル化している．グローバル・スケジューラ内のグループ／プール・サーバ，プライベートスケジューラ，および，グローバル／プライベートメモリマネジャはそれぞれいくつかの関数を提供している．シミュレーションではグループ・サーバが提供する全ての関数の実行時間を同一 ($O_{global\_sche}=2$) とする．また，プール・サーバ／グローバル・メモリマネジャはグループ・サーバによって呼び出されるため，それぞれが提供する関数の実行時間は O_{global_sche} に含まれている．同様にプライベート・スケジューラの実行時間も全て $O_{private\_sche}(=1)$ としている．

また，各サーバ内での実行は排他制御される．すなわち，あるサーバが実行中であれば，そのサーバに発行される他の全ての要求はサーバの実行終了まで待たされる．このようにサーバの実行は逐次化される．

4 スケジューリング方策

ここでは，プロセスの生成／消滅，スレッドの動的生成／消滅などの際に，とるべきアクションについて述べる．アクションは，2.2節で述べたインタフェースを用いて実現される．

また，本論文では，物理メモリ容量は，全プロセス空間を割り当てるだけの十分な容量があると仮定して議論を進める．当然，実際は物理メモリ容量の制限による，ページイン／ページアウトが生じる場合があるが，上記の仮定は，スケジューリングとメモリ管理との協調動作に関する研究の第1段階として設けたものである．はじめから，多くのパラメータを考慮すると，方策による結果の解析が複雑になり，要因の分析が困難になるためである．

4.1 フリープロセッサの分類

NUMAマルチプロセッサでは，フリープロセッサの割り当て方策が1つの鍵となるので，具体的な方策を述べる前にまず，フリープロセッサの分類をしておく．

フリープロセッサは，プロセッサを要求するプロセスの観点から以下の2つの直交軸で分類される．

クラスタによる分類

プロセッサを要求するプロセスとの位置関係により，次の3つに大別される．

- **ホームプロセッサ**：プロセス空間がローディングされているクラスタのプロセッサ．プロセスへのアクセスは，最悪でもローカルアクセスとなる．
- **ローカルプロセッサ**：仮想ページのコピー，移動などによって，プロセス空間の一部が割り当てられているクラスタのプロセッサ．プロセスへのアクセスがローカルアクセスになる可能性がある．
- **リモートプロセッサ**：その他のクラスタのプロセッサ．リモートアクセスのオーバヘッド，または，コピー，移動のオーバヘッドが生じる．

プロセス属性による分類

フリープロセッサのプロセス属性 (プールサーバにより管理される) から，次の3つに大別される．

- **同一属性プロセッサ**：フリーとなる直前に割り当てられていたプロセスと，今プロセッサを要求しているプロセスが同一であるプロセッサ．キャッシュが活用できる可能性がある．
- **空属性プロセッサ**：プロセスの終了によりフリーとなったプロセッサ．キャッシュの活用は見込めない．
- **異属性プロセッサ**：フリーとなる直前に割り当てられていたプロセスと，プロセッサを要求しているプロセスが異なるプロセッサ．本プロセッサは，将来，直前に割り当てられていたプロセスからの要求があり，そのプロセスに対するキャッシュの活用が見込まれる．

4.2 グループサーバ

グループ生成

プロセスが生成されると，グループサーバはグループ生成要求を受けてプロセッサグループを生成する．また，プロセス空間を物理メモリにローディングす

るために，グローバルメモリマネジャに対してメモリマッピング要求を発行する．その後，プライベートスケジューラが起動される．

グループ消滅

グループサーバは，グループ消滅要求を受けとると，該当するプロセッサグループ内の全てのプロセッサを解放し，プロセッサグループを削除する．生じたフリープロセッサを割り当てるプロセスの優先順位に関して，以下の2つの選択肢がある [3]．

1. **実行プロセス優先方式**
2. **待ちプロセス優先方式**

実行プロセスが複数存在する場合の選択順序として，さらに降順方式，昇順方式の2つが存在する [3]．その後，グローバルメモリマネジャに対してメモリマッピングの解除を要求する．

プロセッサ追加

プライベートスケジューラからのプロセッサ追加要求により，プールサーバから割り当てられたプロセッサを該当グループに追加する．

プロセッサ解放

プライベートスケジューラからのプロセッサ解放要求により該当するグループからプロセッサを解放する．解放されたプロセッサの割り当て優先順位に関して，4.2. のグループ消滅の場合と同様に，2つの選択肢が存在する．

4.3 プールサーバ

プールサーバは，フリープロセッサのプロセス属性として，プール内のフリープロセッサがプールに登録される直前に属していたグループの ID (プロセス属性) を記憶している．この情報は，プロセッサの解放時にグループサーバから与えられ，プロセッサとキャッシュの親和性を利用するために用いられる．

プロセッサ割り当て

プロセッサ割り当て要求を受けると，プールサーバはグループサーバから与えられた情報を用いて，プロセッサの割り当てを行なう．選択の順序としては，4.1 節の分類を基に，以下のような選択肢が考えられる．

1. **メモリ管理を考慮しない割当て方式 [3]：**
   (a) 同一属性プロセッサ
   (b) ホームプロセッサ
   (c) 空属性プロセッサ
   (d) その他のプロセッサ
2. **メモリ管理を考慮した方式：**
   (a) ホームプロセッサ
   (b) ローカルプロセッサ
   (c) フリープロセッサの多いクラスタのプロセッサ

プロセッサ登録

該当プロセッサをプロセッサプールに登録する．この時，該当プロセッサが解放されたグループにより，フリープロセッサのプロセス属性を記憶する．

4.4 プライベートスケジューラ

本論文で考慮しているポリシーでは，プロセッサの横取り (preemption) は行なわれない．よって，プロセッサ数変更通知インタフェースは使用しない．各プライベートスケジューラは，プロセス内で FCFS スケジューリングを行なう．

また，スレッドの生成，消滅に対して以下の動作を行なう．

スレッド生成

プライベートスケジューラは，プロセス内の実行中のスレッドと実行待ちスレッドの合計数が，グループに割り当てられたプロセッサ数よりも多い場合，グループサーバに対してプロセッサ追加要求を発行する．

スレッド消滅

スレッドの実行終了により生じたアイドルプロセッサ (idle processor) の取り扱い方策として，以下の2つが存在する [3]．

1. **アイドルプロセッサ保持方式**
2. **アイドルプロセッサ解放方式**

4.5 グローバルメモリマネジャ

メモリマッピング

プロセスが生成されると，グローバルスケジューラからメモリマッピング要求が発行される．これを受けてグローバルメモリマネジャは，ある1つのクラスタの物理メモリに対してプロセス空間のローディングを行なう．このとき，ローディングするクラスタの選定方策に関して，いくつかの選択肢が考えられる．ここではまず，メモリマネジャがスケジューラからの情報を利用するか否かによって，2つに分類する．

1. スケジューラからの情報を必要としないもの
   (a) **巡回方式：**
   プロセスが生成された順番に，各クラスタに順々にローディングする．
   (b) **ランダム選択方式：**
   プロセスが生成する度に，ローディングするクラスタをランダムに決定する．
   (c) **プロセス数最小方式：**
   プロセスの生成時に，ローカルメモリにローディングされているプロセス数が最も少ないクラスタにローディングする．
2. スケジューラからの情報を利用するもの

(d) フリープロセッサ数最大方式：
プロセスの生成時に，フリープロセッサが最も多いクラスタにローディングする．この時点でシステム内にフリープロセッサが存在しなければ，フリープロセッサが生じるまでプロセスはローディングされない．

(e) フリープロセッサが存在するクラスタ内での，プロセス数最小方式：
上記 1.(c) のクラスタに関する判断の候補を，全てのクラスタではなく，フリープロセッサが存在するクラスタに限定したものである．また，2.(d) と同様にこの時点でフリープロセッサが存在しなければ，ローディングは延期される．

この後，生成されたプロセスに対応するプライベートメモリマネージャが起動される．

メモリマッピング解除

プロセスが消滅すると，グローバルスケジューラからメモリマッピング解除要求が発行される．これを受けてグローバルメモリマネジャは，プロセス空間がマッピングされている全ての物理メモリを解放する．

4.6 プライベートメモリマネジャ

メモリ追加

前述したように，プロセス空間全体は，ある 1 つのクラスタの物理メモリにローディングされる．スレッドの実行中，プロセッサによってアクセスされる論理ページがリモートメモリに存在する場合に関して，次の様な選択肢がある．

1. ページのローカルメモリへのキャッシングは行わず，リモートアクセスのままとする．
2. ローカルメモリへキャッシングする．論理ページへのアクセスはローカルアクセスとなる．
このとき，論理ページのキャッシングの方法として，コピー／移動の 2 つの選択肢がある．

メモリ解放

スレッドの実行中，コピーページのコンシステンシ制御や論理ページの移動などによって物理メモリが不要となる場合がある．この場合プライベートメモリマネジャは，不要となった物理メモリを解放するために，メモリ解放要求をグローバルメモリマネジャに対して発行する．

5 評価する方式

ここでは，4節で述べた方策の選択肢の中で，今回比較した選択肢についての整理を行なう．

また，4章で述べた比較方式の内，本研究室におけるこれまでの研究から評価が得られた項目については，選択肢を固定する [3]．すなわち，アイドルプロセッサ処理方式としてはアイドルプロセッサ保持方式，フリープロセッサ処理方式としては実行プロセス優先方式，実行プロセス選択方式としては降順方式をそれぞれ選択する．

5.1 プロセスのローディング

プロセスの生成時に，グローバルメモリマネジャによってプロセス空間の物理メモリへのローディングが行なわれる．今回のシミュレーションでは 4.2節で述べた選択肢の内，以下の 4 方式について比較する．

1. スケジューラからの情報を必要としないもの
   (a) 巡回方式
   (b) プロセス数最小方式
2. スケジューラからの情報を利用するもの
   (c) フリープロセッサ数最大方式
   (d) フリープロセッサが存在する中での，プロセス数最小方式

5.2 フリープロセッサの割り当て

プロセス，スレッドの生成時にプロセッサの割当てが要求された場合，フリープロセッサのグループへの割り当てが行なわれる．本論文では，4.3で述べた，以下の 2 つの方式を比較する．

(A) メモリ管理を考慮しない従来の割当て方式
(B) メモリ管理を考慮した方式

5.3 ページコピー

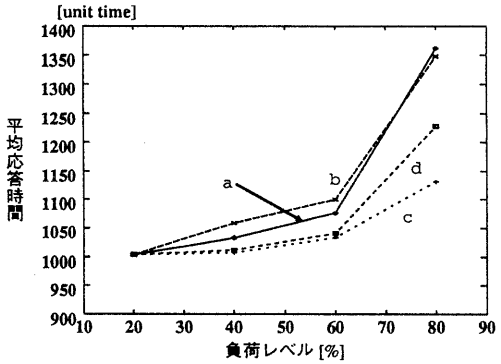
本論文では簡単のため，アクセスする論理ページがローカルメモリにない場合は，読み出し専用ページであればローカルメモリへのページコピーによるキャッシング，読み書きページであればキャッシングは行なわずリモートアクセスのままとする．

6 結果および考察

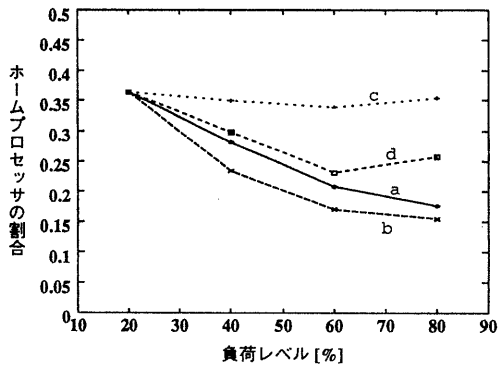
本章では，5節で述べた 2 レベルスケジューリングにおける種々の選択肢を，シミュレーション結果を基に検討する．

6.1 プロセス空間のローディング

まずプロセッサ生成時における，プロセス空間のローディング方式に関する比較を行なう．図 4(1), (2) に比較した各方式における平均応答時間，グループ中のホームプロセッサの割合を示す．図中の a～d は各々

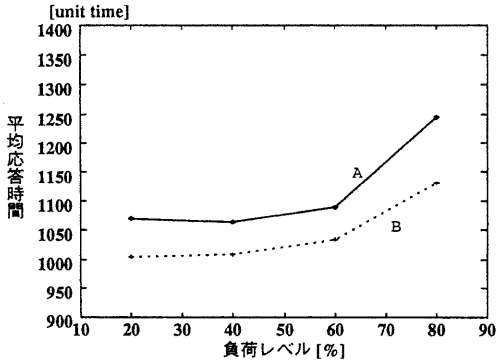


(1) 平均応答時間

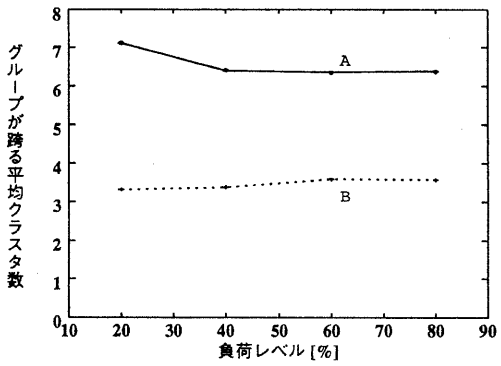


(2) ホームプロセッサの割合

図 4 プロセス空間のローディング



(1) 平均応答時間



(2) 平均クラスタ数

図 5 フリープロセッサ割り当て

5.1節の (a)～(d) に対応する方式である．フリープロセッサの割り当て方式としては，5.2節 (B) メモリ管理を考慮した方式を用いている．

図 4(1) より，スケジューラからの情報としてクラスタのフリープロセッサ数を利用した方式，特に (c) フリープロセッサ数最大方式が良い結果を示していることが分かる．この傾向はシステムの負荷レベルが上昇するに従って顕著になっている．

これは，プロセス生成時にプロセス空間をフリープロセッサ数の少ないクラスタにローディングすることによって，結果的にホームクラスタに多くのプロセッサを確保できていることが原因と考えられる．実際グループ中のホームプロセッサの割合を示す図 4(2) において，フリープロセッサ数最大方式ではほぼ一定した値 (約 35%) を保っている．これにより，メモリアクセスにおけるホームアクセス (プロセス空間がローディングされているクラスタのメモリへのローカルアクセス) の割合が大きくなっていると言える．

同様の理由により，(d) フリープロセッサ数を考慮したプロセス数最小方式が，メモリ管理の持つ情報のみを用いた (a) 静的な巡回方式，(b) プロセス数最小方式より良い結果を示している．従って，メモリマネジャによるプロセス空間のローディングにおいては，スケジューラからの各クラスタのフリープロセッサ数に関する情報提供が有効であるといえる．

6.2 フリープロセッサの割り当て

次にフリープロセッサの割り当てにおける，プロセッサの選択順序に関する選択肢の比較を行なう．図 5(1), (2) に比較した各方式における平均応答時間，グループを構成するプロセッサのまたがるクラスタ数を示す．プロセス空間のローディング方式としては，フリープロセッサ数最大方式を用いている．

図より，メモリ管理を考慮した方式の方が全ての負荷レベルを通して良い結果を示していることが分かる．

この理由としてはまず第一に，フリープロセッサの割り当てに際してプロセス空間がローディングされているクラスタだけでなく，メモリマネジャが持つ情報であるページコピー等によりプロセス空間が割り当てられている量を用いていることが挙げられる．そのため，読み出し専用ページに対するリモートアクセス時

に不必要に多くのクラスタのメモリにページコピーが行なわれることが避けられている．

もう一つの理由としては，プロセッサ割り当てにプロセス空間の割り当てられている量を用いていることに加えて，フリープロセッサの多いクラスタを選択していることにより，必要な数のプロセッサが比較的少数のクラスタから得られていることが考えられる．これもページコピー回数が低く抑えられている原因となっている．

結論としては，フリープロセッサのグループへの割り当てに際しては，メモリマネジャが管理する各クラスタにおけるプロセス空間の割り当て量を用いることが有効である，また，フリープロセッサ数の多いクラスタを優先することも効果が大きいと言える．

7 おわりに

7.1 研究のまとめ

空間分割スケジューリングの一種である2レベルスケジューリングは，大規模マルチプロセッサにおける有望なスケジューリング方式である．本論文では，

- NUMAマルチプロセッサを対象とするメモリ管理を考慮した2レベルスケジューリングの構造とインタフェースの提案．
- シミュレーションによる方策に関する評価．
  - プロセス空間のローディング方式
  - フリープロセッサの割り当て方式

を行なった．その結果NUMAマルチプロセッサを対象とした2レベルスケジューラとメモリマネジャの情報交換に関して，次のような結論に達した．

- プロセス空間のローディング方式にはスケジューラからの情報提供を受けたフリープロセッサ数最大方式が優れている．
- フリープロセッサの割り当て方式としてはメモリ管理を考慮した方式が優れている．

7.2 今後の課題

今後の課題としては以下のようなものが挙げられる．

- スケジューラとメモリマネジャの情報交換についてのさらなる選択肢についての評価．
  今回研究の第一ステップとして，モデル化の段階で幾つかの仮定の上でシミュレーションを行なった．今後より現実的なモデルを用いてシミュレーションを行なった場合，さらに効果的な情報交換の必要性が生じてくると考えられる．
- 種々のプロセスモデルに対する方式の評価．
  今回のシミュレーションでは，唯一つのプロセスグラフに固定して実験を行なった．しかし，一般的な結論を得るためには幾つかのプロセスモデルに対して方式の比較を行なう必要がある．
- I/Oのモデル化．
  プライベートスケジューラによるスレッド間のスケジューリングでは，I/Oリクエスト等が起こった場合にスレッド切替を起こすことが通常行なわれると考えられるが，今回のシミュレーションではI/Oを無視したモデル上での評価となっている．今後さらにＩ／Ｏを含めて本スケジューリングを評価することは，最も重要な課題であるといえる．
- アプリケーション依存と非依存の分離
  メモリアクセスの挙動は，アプリケーションに強く依存する．従って，メモリ管理を考慮したスケジューリングにおいて，アプリケーションに依存するものと非依存のものとを切り出す必要がある．

参考文献

[1] 甲斐久淳，桑山雅行，最所圭三，福田晃：共有メモリ型マルチプロセッサにおけるスケジューリング方式の評価ープロセッサ・グループによる2レベルスケジュリングー，情報処理学会「コンピュータシステム・シンポジウム」，pp.77-84(1992年10月)．

[2] 藤木亮介，甲斐久淳，福田晃：NUMAマルチプロセッサにおける2レベルスケジューリング，情報処理学会システム・ソフトウェアとオペレーティング・システム研究会，pp.61-15（1993年8月）．

[3] 藤木亮介，甲斐久淳，福田晃：ＮＵＭＡマルチプロセッサにおける2レベルスケジューリングアルゴリズムの評価，情報処理学会「コンピュータシステム・シポジウム」，pp.67-74（1993年10月）．

[4] Gupta, A., Tucker, A., and Urushibara, S.: The Impact of Operating System Scheduling and Synchronization Methods on the Performance of Parallel Applications, Proc. the 1991 ACM SIGMETRICS Conf. on Measurement and Modelling of Computer Systems, pp.120-132(1991).

[5] Ousterhout, J.K.: Scheduling Techniques for Concurrent Systems, Proc. 3rd Int'l Conf. on Distributed Computing Systems, pp.22-30(1982).

[6] Zho, S. and Brecht, T.: Processor Pool-Based Scheduling for Large-Scale NUMA Multiprocessors, Proc. the 1991 ACM SIGMETRICS Conf. Measurement and Modelling of Computer Systems, pp.133-142(1991).

[7] M., Evangelos, C., Crovella, D., Prakash, D., Cezary, and L., Thomas: The Effects of Multiprogramming on Barrier Synchronization, Proc. the third IEEE Symposium on Parallel and Distributed Processing, pp.662-669(1991).

[8] Bolosky, W.J., Fitzgerald, R.P. and Scott, M.L.: Simple But Effective Techniques for NUMA Memory Management, Proc. the Twelfth ACM Symposium on Operating Systems Principles, pp.19-31(1991).